



Many-core Design for Data-Flow Execution

Using the SWitches Prototype Implementation

Andreas Diavastos

diavastos@cs.ucy.ac.cy

Advisor: **Pedro Trancoso**



Department of Computer Science, University of Cyprus

CASPER Group: Computer Architecture, Systems and Performance Evaluation Research Group

Introduction

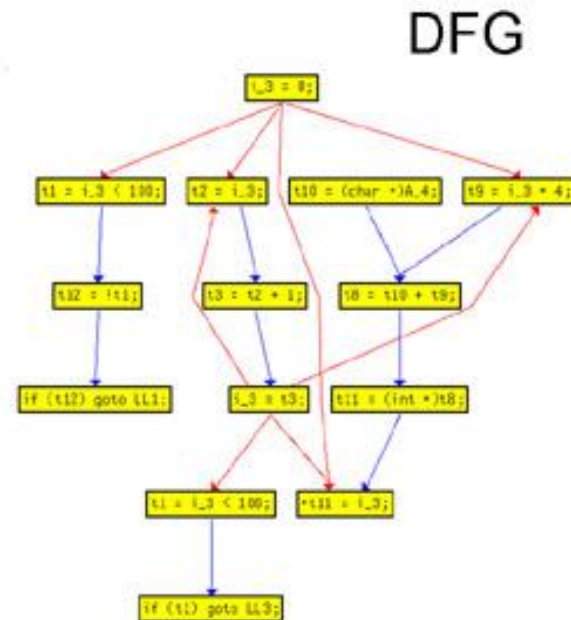
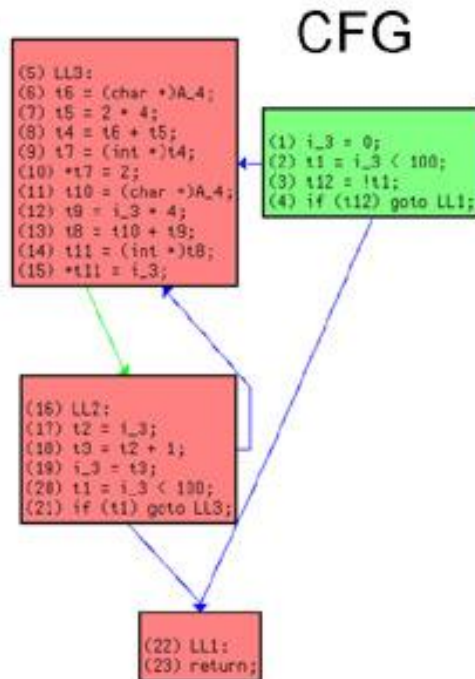
- **Multi-cores:**
 - 2 – 16 cores
 - Designed for **small-scale parallelism & for sequential performance**
- **Many-cores:**
 - 10s – 100s – 1000s cores
 - **Design for large-scale parallelism ONLY!**
 - **Sequential programs will run much slower!**
 - If they can execute



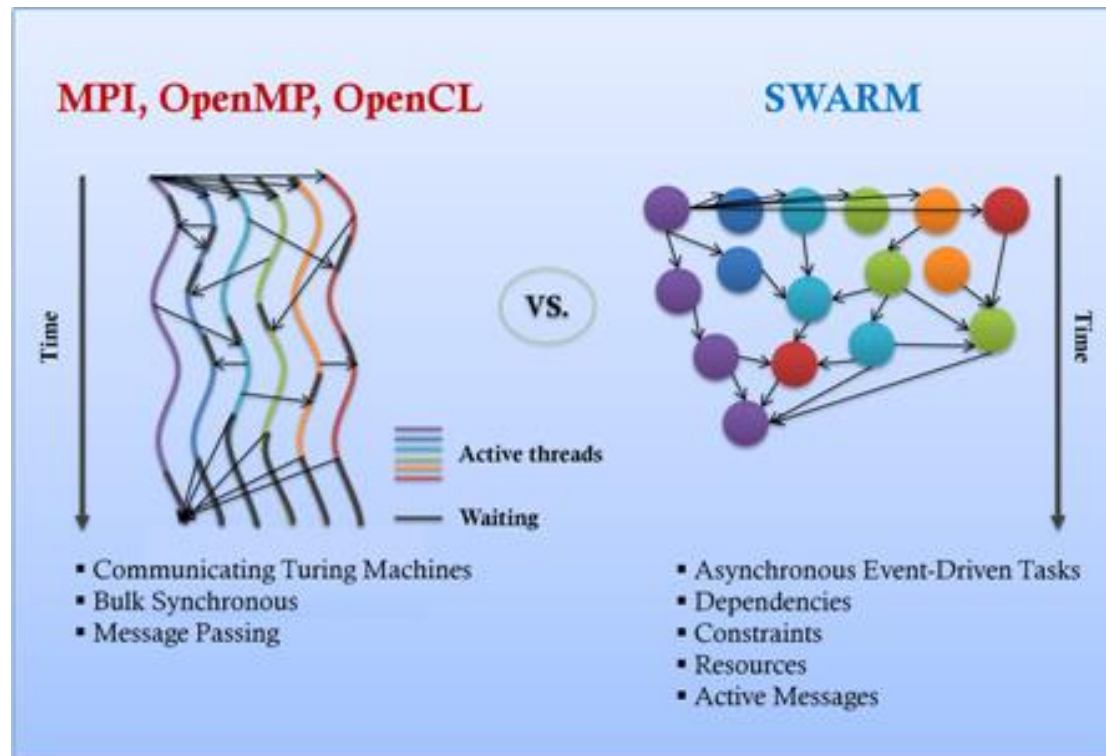
Introduction

Control-flow

Data-flow

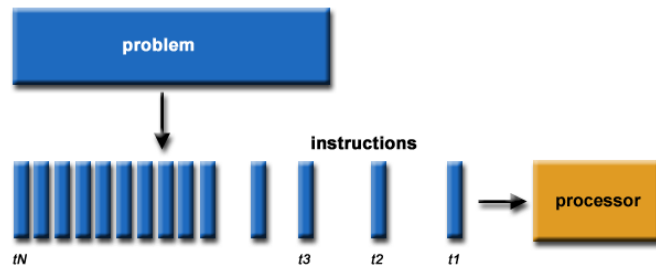


Introduction

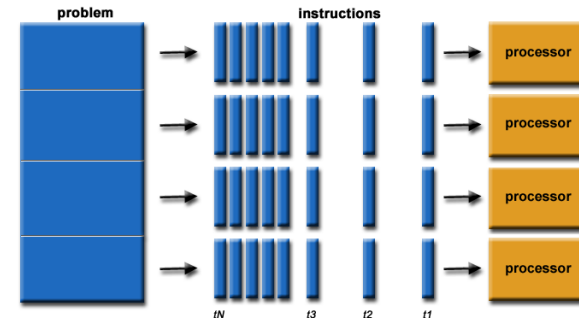


What is parallelism?

Serial Processor



Parallel Processor



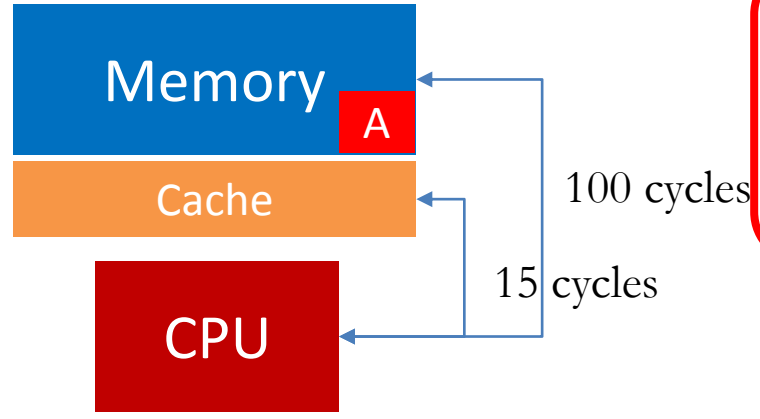
- Many processors executing instructions from the same program in parallel
- Instruction execution time is the same
- **Program execution time is less!!**





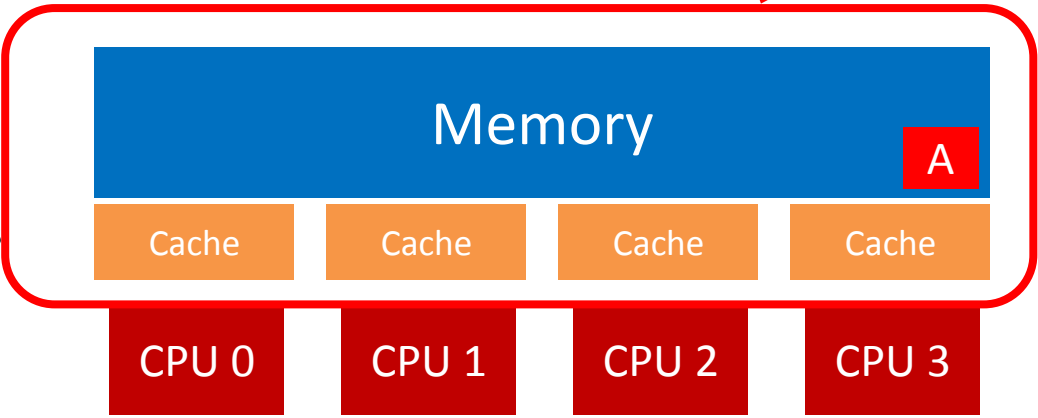
What is cache-coherence?

Serial Processor



$$A = 20$$

Parallel Processor



$$B = A / 2$$

$$B = A * 2$$

$$A = 10$$

$$A = A + 1$$

- $B = A / 2$
- $A = 10$
- $A = A + 1$
- $B = A * 2$

Motivation

- Many-core processing for increased parallelism
 - HPC systems already include many-cores
- More parallelism → More performance
- Many-cores today (and tomorrow?)
 - Cache-coherent Shared Memory
 - Can they scale?
 - Distributed Memory Clustered
 - Are they efficient for fine-grain parallelism?
- Software Parallel Programming Systems
 - Shared-memory
 - Good on multi-cores, **not evaluated on many-cores**
 - **Need cache-coherence!**
 - Distributed-memory
 - Good for large-scale distributed systems
 - **Programming is becoming an impossible task**



We need:

- Scalable Hardware:
 - A **non-coherent shared-memory** many-core
- Scalable Software:
 - Easy **programming**
 - Non-blocking execution
 - Doesn't need cache-coherence support
 - Exploits large amounts of **fine-grain parallelism**



Scale performance on non-coherent
shared-memory many-core processors



Expected Contributions:



- Non-coherent shared-memory many-core design
 - Simulation-based
 - With 100s of cores
- A software parallel programming system
 - Based on data-flow execution
 - Scale performance to 100s of cores
 - Validate to real hardware
 - Support for conventional programming APIs
 - OpenMP



Outline

- Motivation
- Contributions
- Related Work
 - Data-flow-based Programming Systems
 - DDM Programming Systems
- SWitches Programming System
- Preliminary Work
- Roadmap – Timeline



Outline

- Motivation
- Contributions
- Related Work
 - Data-flow-based Programming Systems
 - DDM Programming Systems
- SWitches Programming System
- Preliminary Work
- Roadmap – Timeline



Data-flow Programming Systems

Characteristics of Dataflow Implementations	OmpSs	Triggered Instructions	Serialization Sets	OpenDF	DTT / CDTT	SEED	Statically Sequential	WaveScalar	SWARM	Intel TBB	CnC	Maxeler
Implementation (Software / Hardware)	Software	Hardware	Software	Hardware	Software	Hardware	Software	Hardware	Software	Software	Software	Hardware
Scheduling Policy (Static / Dynamic)	Dynamic	Dynamic	Dynamic	-	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic - uses Intel TBB	Static
Memory Model (Shared / Private)	Shared / GPU	Shared	Shared	-	Shared	Shared	Shared	Shared	Shared / Distributed	Shared	Shared	-
Needs cache-coherency	Yes	No	Yes	-	-	-	Yes	Yes	Yes	Yes	Yes	-
Number of cores/threads tested	24	32	32	-	runtime	-	32	128 Simulation	24	8	8	-
Max Speedup achieved	depends on application	22	16	-	1.46 / single core	-	16	83	8	8	8	230
How dependencies are expressed	directives in(), out(), inout()	Inserted Triggers	writable & read_only data variables	-	macro-based triggers	-	functions, shared objects, read/write sets, sequential segments	tokens/tags	C-macros API to represent Codelets	macro-API / explicit task dependencies	Inputs (gets()), Outputs (puts())	Kernels & Managers, Input & Output vars.
Programming Language	C / C++	C / C++	C++	CAL Dataflow	C / C++	-	C++	Supports imperative languages	C	C++	C++, Java, .NET, Haskell	Java with MaxCompiler
Contributions	Single programming model for homogeneous & heterogeneous architectures	<ul style="list-style-type: none"> Spatial accel. w. 8x greater area-performa. than GPPs Less instr. in critical path over PC-based spatial architectures 	Prometheus C++ library that implements Serialization Sets	Implementation of MPEG RVC decoder on CAL dataflow model	<ul style="list-style-type: none"> High redundant code in apps Data-triggered threads Architectural support for DTT 	<ul style="list-style-type: none"> High memory parallelism, instruction parallelism and branch unpredictability is highly profitable for dataflow execution 	C++ Software runtime library	<ul style="list-style-type: none"> New dataflow ISA Less area occupied for logic More performance/area 	<ul style="list-style-type: none"> Unified single-, multi-node interface, transparent to the programmer 	<ul style="list-style-type: none"> Parallel algorithms and data structures Scalable memory allocation and task scheduling 	<ul style="list-style-type: none"> CnC semantics w proof of determinism Exploit several types of parallelism Performance scalability 	Less silicon area, less power consumption: No instructions/instr. decode logic No branch predict. No GP caches
Conference / Date	PPL 2011	ISCA 2013	PPoPP 2009	ACM SIGARCH 2008	HPCA 2009	ISCA 2015	MICRO 2011	ACM Transactions 2007	2013	2007	2011	ACM SIGARCH 2011
Notes	<ul style="list-style-type: none"> Based on StarSs and OpenMP Builds graph at runtime Task-dependency graph Each task is executed once 	<ul style="list-style-type: none"> Using scratchpad memory Dynamic Instruction reordering FPGA Implem. New ISA extentions 	<ul style="list-style-type: none"> Supports all types of parallelism (Data, Task, Pipeline, Embarrassing) Not better than OpenMP or Pthreads 	Produces VHDL code	<ul style="list-style-type: none"> Remove redundant unnecessary computation Thread generation based on address data changes 	Switching execution from OOO to dataflow at runtime based on application needs	<ul style="list-style-type: none"> Based on Prometheus runtime of Serialization Sets This is the same as Serialization Sets 	<ul style="list-style-type: none"> Only measure the time for the parallel phase 	<ul style="list-style-type: none"> Very difficult programming Work stealing across nodes and threads One scheduler on each thread/node, managing several codelets Overhead of runtime observed for fine-grain scheduling 	<ul style="list-style-type: none"> Rich feature set for general purpose parallelism Data-dependency graph Each task can execute multiple times 	<ul style="list-style-type: none"> Comparison only with Pthreads Different system for each application CnC is a model that uses other runtime systems 	-



Data-flow Programming Systems

Characteristics of Dataflow Implementations	OmpSs	Triggered Instructions	Serialization Sets	OpenDF	DTT / CDTT	SEED	Statically Sequential	WaveScalar	SWARM	Intel TBB	CnC	Maxeler
Implementation (Software / Hardware)	Software	Hardware	Software	Hardware	Software	Hardware	Software	Hardware	Software	Software	Software	Hardware
Scheduling Policy (Static / Dynamic)	Dynamic	Dynamic	Dynamic	-	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic - uses Intel TBB	Static
Memory Model (Shared / Private)	Shared / GPU	Shared	Shared	-	Shared	Shared	Shared	Shared	Shared / Distributed	Shared	Shared	-
Needs cache-coherency	Yes	No	Yes	-	-	-	Yes	Yes	Yes	Yes	Yes	-
Number of cores/threads tested	24	32	32	-	runtime	-	32	128 Simulation	24	8	8	-
Max Speedup achieved	depends on application	22	16	-	1.46 / single core	-	16	83	8	8	8	230
How dependencies are expressed	directives in(), out(), inout()	Inserted Triggers	writable & read_only data variables	-	macro-based triggers	-	functions, shared objects, read/write sets, sequential segments	tokens/tags	C-macros API to represent Codelets	macro-API / explicit task dependencies	Inputs (gets()), Outputs (puts())	Kernels & Managers, Input & Output vars.
Programming Language	C / C++	C / C++	C++	CAL Dataflow	C / C++	-	C++	Supports imperative languages	C	C++	C++, Java, .NET, Haskell	Java with MaxCompiler
Contributions	Single programming model for homogeneous & heterogeneous architectures	<ul style="list-style-type: none"> Spatial accel. w. 8x greater area-performa. than GPPs Less instr. in critical path over PC-based spatial architectures 	Prometheus C++ library that implements Serialization Sets	Implementation of MPEG RVC decoder on CAL dataflow model	<ul style="list-style-type: none"> High redundant code in apps Data-triggered threads Architectural support for DTT 	<ul style="list-style-type: none"> High memory parallelism, instruction parallelism and branch unpredictability is highly profitable for dataflow execution 	C++ Software runtime library	<ul style="list-style-type: none"> New dataflow ISA Less area occupied for logic More performance/area 	<ul style="list-style-type: none"> Unified single-, multi-node interface, transparent to the programmer 	<ul style="list-style-type: none"> Parallel algorithms and data structures Scalable memory allocation and task scheduling 	<ul style="list-style-type: none"> CnC semantics w proof of determinism Exploit several types of parallelism Performance scalability 	Less silicon area, less power consumption: No instructions/instr. decode logic No branch predict. No GP caches
Date	PPL 2011	ISCA 2013	PPoPP 2009	ACM SIGARCH 2008	HPCA 2009	ISCA 2015	MICRO 2011	ACM Transactions 2007	2013	2007	2011	ACM SIGARCH 2011
Notes	<ul style="list-style-type: none"> Based on StarSs and OpenMP Builds graph at runtime Task-dependency graph Each task is executed once 	<ul style="list-style-type: none"> Using scratchpad memory Dynamic Instruction reordering FPGA Implem. New ISA extentions 	<ul style="list-style-type: none"> Supports all types of parallelism (Data, Task, Pipeline, Embarrassing) Not better than OpenMP or Pthreads 	Produces VHDL code	<ul style="list-style-type: none"> Remove redundant unnecessary computation Thread generation based on address data changes 	Switching execution from OOO to dataflow at runtime based on application needs	<ul style="list-style-type: none"> Based on Prometheus runtime of Serialization Sets This is the same as Serialization Sets 	<ul style="list-style-type: none"> Only measure the time for the parallel phase 	<ul style="list-style-type: none"> Very difficult programming Work stealing across nodes and threads One scheduler on each thread/node, managing several codelets Overhead of runtime observed for fine-grain scheduling 	<ul style="list-style-type: none"> Rich feature set for general purpose parallelism Data-dependency graph Each task can execute multiple times 	<ul style="list-style-type: none"> Comparison only with Pthreads Different system for each application CnC is a model that uses other runtime systems 	-



Data-flow Programming Systems

Characteristics of Dataflow Implementations	OmpSs	Serialization Sets	DTT / CDTT	Statically Sequential	SWARM	Intel TBB	CnC
Implementation (Software / Hardware)	Software	Software	Software	Software	Software	Software	Software
Scheduling Policy (Static / Dynamic)	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic - uses Intel TBB
Memory Model (Shared / Private)	Shared / GPU	Shared	Shared	Shared	Shared / Distributed	Shared	Shared
Needs cache-coherency	Yes	Yes	-	Yes	Yes	Yes	Yes
Number of cores/threads tested	24	32	runtime	32	24	8	8
Max Speedup achieved	depends on application	16	1.46 / single core	16	8	8	8
How dependencies are expressed	directives in(), out(), inout()	writable & read_only data variables	macro-based triggers	functions, shared objects, read/write sets, sequential segments	C-macros API to represent Codelets	macro-API / explicit task dependencies	Inputs (gets()), Outputs (puts())
Programming Language	C / C++	C++	C / C++	C++	C	C++	C++, Java, .NET, Haskell
Contributions	Single programming model for homogeneous & heterogeneous architectures	Prometheus C++ library that implements Serialization Sets	<ul style="list-style-type: none"> High redundant code in apps Data-triggered threads Architectural support for DTT 	C++ Software runtime library	<ul style="list-style-type: none"> Unified single-, multi-node interface, transparent to the programmer 	<ul style="list-style-type: none"> Parallel algorithms and data structures Scalable memory allocation and task scheduling 	<ul style="list-style-type: none"> CnC semantics w proof of determinism Exploit several types of parallelism Performance scalability
Date	PPL 2011	PPoPP 2009	HPCA 2009	MICRO 2011	2013	2007	ACM SIGARCH 2011
Notes	<ul style="list-style-type: none"> Based on StarSs and OpenMP Builds graph at runtime Task-dependency graph <ul style="list-style-type: none"> Each task is executed once 	<ul style="list-style-type: none"> Supports all types of parallelism (Data, Task, Pipeline, Embarrassing) Not better than OpenMP or Pthreads 	<ul style="list-style-type: none"> Remove redundant unnecessary computation Thread generation based on address data changes 	<ul style="list-style-type: none"> Based on Prometheus runtime of Serialization Sets This is the same as Serialization Sets 	<ul style="list-style-type: none"> Very difficult programming Work stealing across nodes and threads One scheduler on each thread/node, managing several codelets Overhead of runtime observed for fine-grain scheduling 	<ul style="list-style-type: none"> Rich feature set for general purpose parallelism Data-dependency graph <ul style="list-style-type: none"> Each task can execute multiple times 	<ul style="list-style-type: none"> Comparison only with Pthreads Different system for each application CnC is a model that uses other runtime systems



Data-flow Programming Systems

Characteristics of Dataflow Implementations	OmpSs	Serialization Sets	DTT / CDTT	Statically Sequential	SWARM	Intel TBB	CnC
Implementation (Software / Hardware)	Software	Software	Software	Software	Software	Software	Software
Scheduling Policy (Static / Dynamic)	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic - uses Intel TBB
Memory Model (Shared / Private)	Shared / GPU	Shared	Shared	Shared	Shared / Distributed	Shared	Shared
Needs cache-coherency	Yes	Yes	-	Yes	Yes	Yes	Yes
Number of cores/threads tested	24	32	runtime	32	24	8	8
Max Speedup achieved	depends on application	16	1.46 / single core	16	8	8	8
How dependencies are expressed	directives in(), out(), inout()	writable & read_only data variables	macro-based triggers	functions, shared objects, read/write sets, sequential segments	C-macros API to represent Codelets	macro-API / explicit task dependencies	Inputs (gets()), Outputs (puts())
Programming Language	C / C++	C++	C / C++	C++	C	C++	C++, Java, .NET, Haskell
Contributions	Single programming model for homogeneous & heterogeneous architectures	Prometheus C++ library that implements Serialization Sets	<ul style="list-style-type: none"> High redundant code in apps Data-triggered threads Architectural support for DTT 	C++ Software runtime library	<ul style="list-style-type: none"> Unified single-, multi-node interface, transparent to the programmer 	<ul style="list-style-type: none"> Parallel algorithms and data structures Scalable memory allocation and task scheduling 	<ul style="list-style-type: none"> CnC semantics w proof of determinism Exploit several types of parallelism Performance scalability
Date	PPL 2011	PPoPP 2009	HPCA 2009	MICRO 2011	2013	2007	ACM SIGARCH 2011
Notes	<ul style="list-style-type: none"> Based on StarSs and OpenMP Builds graph at runtime Task-dependency graph <ul style="list-style-type: none"> Each task is executed once 	<ul style="list-style-type: none"> Supports all types of parallelism (Data, Task, Pipeline, Embarrassing) Not better than OpenMP or Pthreads 	<ul style="list-style-type: none"> Remove redundant unnecessary computation Thread generation based on address data changes 	<ul style="list-style-type: none"> Based on Prometheus runtime of Serialization Sets This is the same as Serialization Sets 	<ul style="list-style-type: none"> Very difficult programming Work stealing across nodes and threads One scheduler on each thread/node, managing several codelets Overhead of runtime observed for fine-grain scheduling 	<ul style="list-style-type: none"> Rich feature set for general purpose parallelism Data-dependency graph <ul style="list-style-type: none"> Each task can execute multiple times 	<ul style="list-style-type: none"> Comparison only with Pthreads Different system for each application CnC is a model that uses other runtime systems



Data-flow Programming Systems

Characteristics of Dataflow Implementations	OmpSs	SWARM	Intel TBB
Implementation (Software / Hardware)	Software	Software	Software
Scheduling Policy (Static / Dynamic)	Dynamic	Dynamic	Dynamic
Memory Model (Shared / Private)	Shared / GPU	Shared / Distributed	Shared
Needs cache-coherency	Yes	Yes	Yes
Number of cores/threads tested	24	24	8
Max Speedup achieved	depends on application	8	8
How dependencies are expressed	directives in(), out(), inout()	C-macros API to represent Codelets	macro-API / explicit task dependencies
Programming Language	C / C++	C	C++
Contributions	Single programming model for homogeneous & heterogeneous architectures	<ul style="list-style-type: none"> Unified single-, multi-node interface, transparent to the programmer 	<ul style="list-style-type: none"> Parallel algorithms and data structures Scalable memory allocation and task scheduling
Date	2011	2013	2007
Notes	<ul style="list-style-type: none"> Based on StarSs and OpenMP Builds graph at runtime Task-dependency graph Each task is executed once 	<ul style="list-style-type: none"> Difficult programming Work stealing across nodes and threads One scheduler on each thread/node Overhead of runtime observed for fine-grain scheduling 	<ul style="list-style-type: none"> Rich feature set for general purpose parallelism Data-dependency graph Each task can execute multiple times



Data-flow Programming Systems

Characteristics of Dataflow Implementations	OmpSs	SWARM	Intel TBB
Implementation (Software / Hardware)	Software	Software	Software
Scheduling Policy (Static / Dynamic)	Dynamic	Dynamic	Dynamic
Memory Model (Shared / Private)	Shared / GPU	Shared / Distributed	Shared
Needs cache-coherency	Yes	Yes	Yes
Number of cores/threads tested	24	24	8
Max Speedup achieved	depends on application	8	8
How dependencies are expressed	directives in(), out(), inout()	C-macros API to represent Codelets	macro-API / explicit task dependencies
Programming Language	C / C++	C	C++
Contributions	Single programming model for homogeneous & heterogeneous architectures	<ul style="list-style-type: none"> Unified single-, multi-node interface, transparent to the programmer 	<ul style="list-style-type: none"> Parallel algorithms and data structures Scalable memory allocation and task scheduling
Date	2011	2013	2007
Notes	<ul style="list-style-type: none"> Based on StarSs and OpenMP Builds graph at runtime Task-dependency graph Each task is executed once 	<ul style="list-style-type: none"> Difficult programming Work stealing across nodes and threads One scheduler on each thread/node Overhead of runtime observed for fine-grain scheduling 	<ul style="list-style-type: none"> Rich feature set for general purpose parallelism Data-dependency graph Each task can execute multiple times



Data-flow Programming Systems

Characteristics of Dataflow Implementations	OmpSs	SWARM	Intel TBB
Implementation (Software / Hardware)	Software	Software	Software
Scheduling Policy (Static / Dynamic)	Dynamic	Dynamic	Dynamic
Memory Model (Shared / Private)	Shared / GPU	Shared / Distributed	Shared
Needs cache-coherency	Yes	Yes	Yes
Number of cores/threads tested	24	24	8
Max Speedup achieved	depends on application	8	8
How dependencies are expressed	directives in(), out(), inout()	C-macros API to represent Codelets	macro-API / explicit task dependencies
Programming Language	C / C++	C	C++
Contributions	Single programming model for homogeneous & heterogeneous architectures	<ul style="list-style-type: none"> Unified single-, multi-node interface, transparent to the programmer 	<ul style="list-style-type: none"> Parallel algorithms and data structures Scalable memory allocation and task scheduling
Date	2011	2013	2007
Notes	<ul style="list-style-type: none"> Based on StarSs and OpenMP Builds graph at runtime Task-dependency graph Each task is executed once 	<ul style="list-style-type: none"> Difficult programming Work stealing across nodes and threads One scheduler on each thread/node Overhead of runtime observed for fine-grain scheduling 	<ul style="list-style-type: none"> Rich feature set for general purpose parallelism Data-dependency graph Each task can execute multiple times



Data-flow Programming Systems

Characteristics of Dataflow Implementations	OmpSs	SWARM	Intel TBB
Implementation (Software / Hardware)	Software	Software	Software
Scheduling Policy (Static / Dynamic)	Dynamic	Dynamic	Dynamic
Memory Model (Shared / Private)	Shared / GPU	Shared / Distributed	Shared
Needs cache-coherency	Yes	Yes	Yes
Number of cores/threads tested	24	24	8
Max Speedup achieved	depends on application	8	8
How dependencies are expressed	directives in(), out(), inout()	C-macros API to represent Codelets	macro-API / explicit task dependencies
Programming Language	C / C++	C	C++
Contributions	Single programming model for homogeneous & heterogeneous architectures	<ul style="list-style-type: none"> Unified single-, multi-node interface, transparent to the programmer 	<ul style="list-style-type: none"> Parallel algorithms and data structures Scalable memory allocation and task scheduling
Date	2011	2013	2007
Notes	<ul style="list-style-type: none"> Based on StarSs and OpenMP Builds graph at runtime Task-dependency graph Each task is executed once 	<ul style="list-style-type: none"> Difficult programming Work stealing across nodes and threads One scheduler on each thread/node Overhead of runtime observed for fine-grain scheduling 	<ul style="list-style-type: none"> Rich feature set for general purpose parallelism Data-dependency graph Each task can execute multiple times



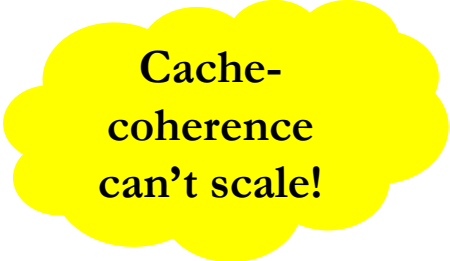
Data-flow Programming Systems

Characteristics of Dataflow Implementations	OmpSs	SWARM	Intel TBB
Implementation (Software / Hardware)	Software	Software	Software
Scheduling Policy (Static / Dynamic)	Dynamic	Dynamic	Dynamic
Memory Model (Shared / Private)	Shared / GPU	Shared / Distributed	Shared
Needs cache-coherency	Yes	Yes	Yes
Number of cores/threads tested	24	24	8
Max Speedup achieved	depends on application	8	8
How dependencies are expressed	directives in(), out(), inout()	C-macros API to represent Codelets	macro-API / explicit task dependencies
Programming Language	C / C++	C	C++
Contributions	Single programming model for homogeneous & heterogeneous architectures	<ul style="list-style-type: none"> Unified single-, multi-node interface, transparent to the programmer 	<ul style="list-style-type: none"> Parallel algorithms and data structures Scalable memory allocation and task scheduling
Date	2011	2013	2007
Notes	<ul style="list-style-type: none"> Based on StarSs and OpenMP Builds graph at runtime Task-dependency graph Each task is executed once 	<ul style="list-style-type: none"> Difficult programming Work stealing across nodes and threads One scheduler on each thread/node Overhead of runtime observed for fine-grain scheduling 	<ul style="list-style-type: none"> Rich feature set for general purpose parallelism Data-dependency graph Each task can execute multiple times



Conclusions on Data-flow Systems

- Dynamic scheduling ^(all)
 - Adds runtime overhead
- Centralized Runtime ^(OmpSs, TBB)
 - Single-point of access
- Scale up to 24 cores ^(all)
 - None is tested to more cores
- Runtime Dependency resolution ^(OmpSs)
 - Adds runtime overhead
- Difficult Programming ^(SWARM, TBB)
 - Macro-based programming
 - Programmer responsible for dependencies, updates, etc.
- Shared memory ^(OmpSs, TBB, SWARM)
 - But need hardware support for cache-coherence



Cache-coherence can't scale!

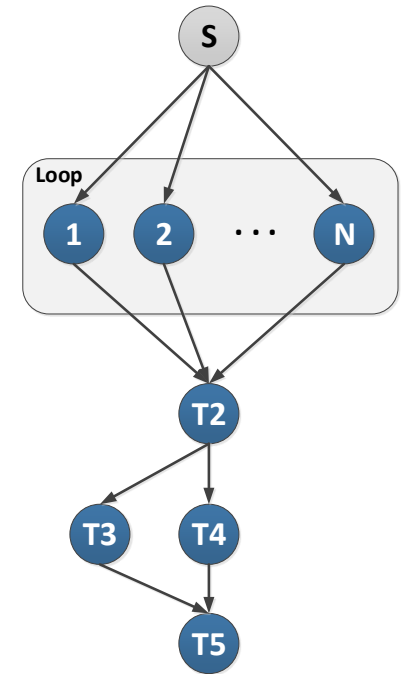


Wait!



The DDM Model

- Data-Driven Multithreading Model
 - Data-Flow execution
 - Thread-based
 - Synchronization (Data-flow) graph
 - No locks
 - No barriers
 - Only dependencies
 - Control-Flow execution within a thread
 - Exploit inherent architecture & compiler optimizations



DDM Programming Systems

Characteristics of Dataflow Implementations	D ² NOW	TFlux	DDM-VM _c	DDM-VM _s	DDM-VM _d	DDM _{FPGA}	TFluxSCC	TFluxTM	
TSU Implementation (Software / Hardware)	Hardware	Software	Hardware	Software	Software	Software	Hardware	Software	Software
Scheduling Policy (Static / Dynamic)	-	Static	Static & Dynamic	Static & Dynamic	Static	Static & Dynamic	Static	Static	
Memory Model (Shared / Private)	Distributed Shared Memory	Shared	Private	Shared	Private	Shared	Shared Data / Distributed Runtime	Shared	
Needs cache-coherency	-	Yes	No	Yes	Yes	Yes	No	Yes	
Number of cores/threads tested	32	6	27	6 SPEs + 1 PPE	12	12 cores x 2 nodes	8	48	12
Max Speedup achieved	26	5.9	25	Almost Linear	9.6	9.6 (SMT) / 16 (Distr)	7.96	48	6.2
How dependencies are expressed	macros	Directives	macros	macros	macros	macros	Directives	Directives	
Contributions	<ul style="list-style-type: none"> • CacheFlow • 1st DDM Simulated Hardware Distributed 	<ul style="list-style-type: none"> • 1st SMP Software implementation • 1st full system simulation • Complete & Portable • Directive-based programming 	<ul style="list-style-type: none"> • 1st heterog. Implement. • Software CacheFlow Implement. 	<ul style="list-style-type: none"> • 1st Software CacheFlow 	<ul style="list-style-type: none"> • 1st software distributed 	<ul style="list-style-type: none"> • 1st hardware DDM FPGA 	<ul style="list-style-type: none"> • 1st many-core software DDM 	<ul style="list-style-type: none"> • 1st integrat. of a DDM implementation with another model 	
Date	2000	2008	2011	2010	2013	2014	2014 & 2015	2011 & 2015	



Conclusions on DDM Systems

- Centralized Runtime (all – except TFluxSCC)
 - Single-point of communication
- **Need hardware cache-coherence** (all – except TFluxSCC)
- Global SG (all – except TFluxSCC)
 - Requires protection (e.g. locking)
 - Not scalable!
 - TFluxSCC uses one SG instance for every core
 - Memory expensive!

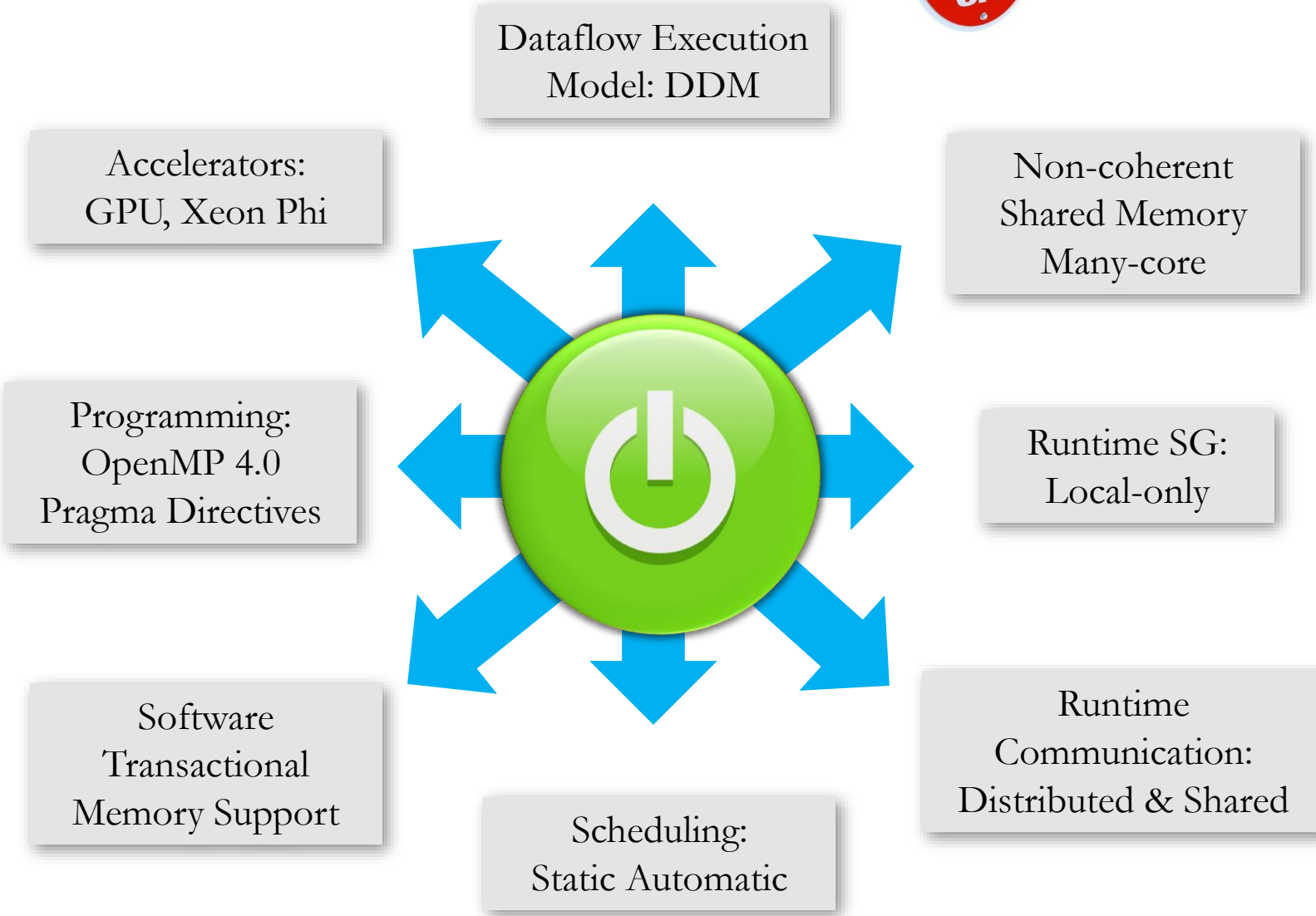


Outline

- Motivation
- Contributions
- Related Work
 - Data-flow-based Programming Systems
 - DDM Programming Systems
- **SWitches Programming System**
- Preliminary Work
- Roadmap – Timeline



SWitches Programming System

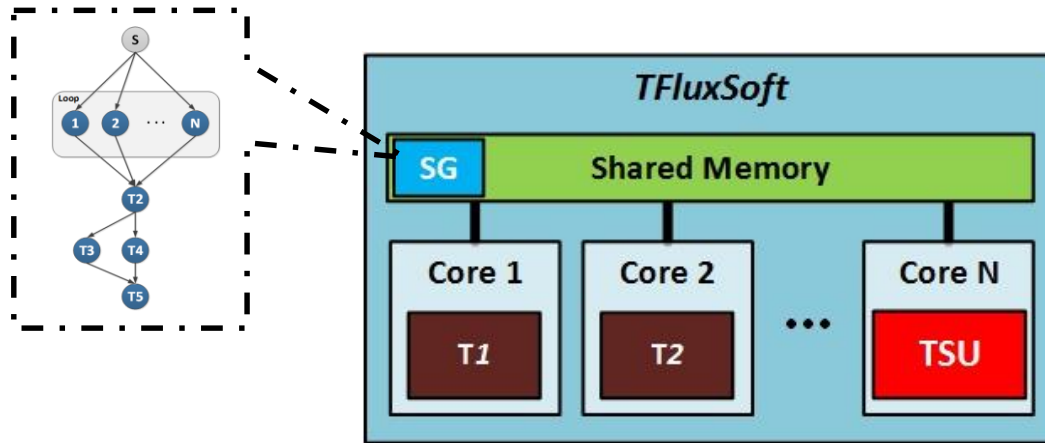


Outline

- Motivation
- Contributions
- Related Work
 - Data-flow-based Programming Systems
 - DDM Programming Systems
- SWitches Programming System
- Preliminary Work
- Roadmap – Timeline



From TFlux to SWitches - TFluxSoft

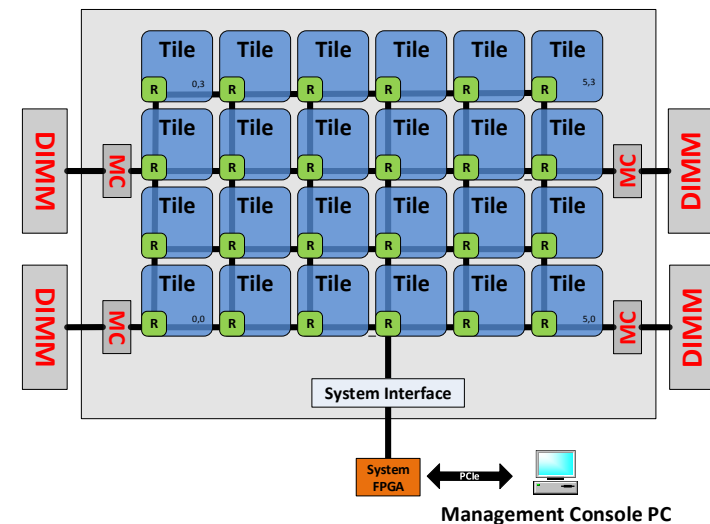
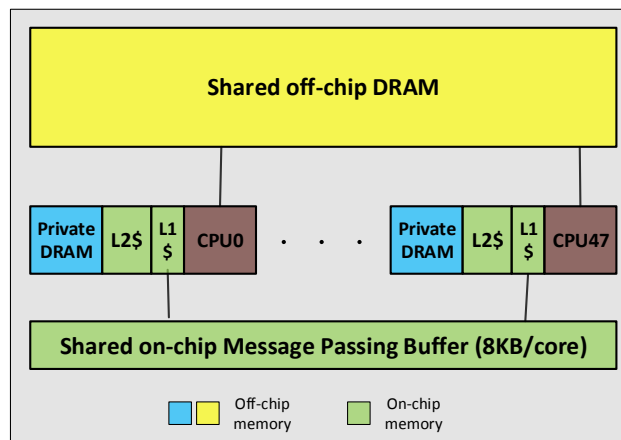


- Software implementation of the DDM model
 - For shared-memory multi-cores
 - Shared memory for updates exchange
- Global Synchronization Graph (SG)
 - Locking is required
 - Cache-coherence is required!
- Centralized scheduling unit (TSU)
 - Single-point of thread updates



Intel Single-chip Cloud Computing

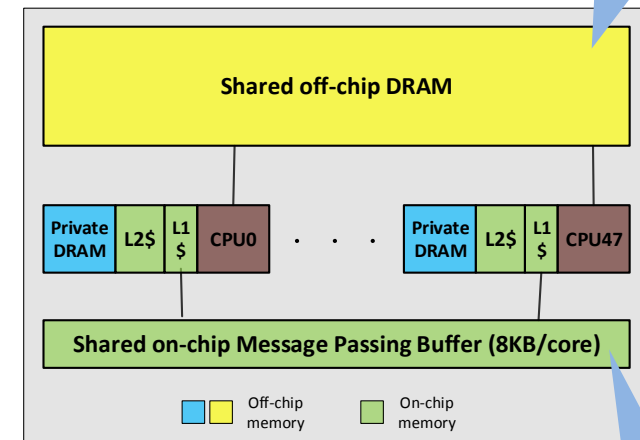
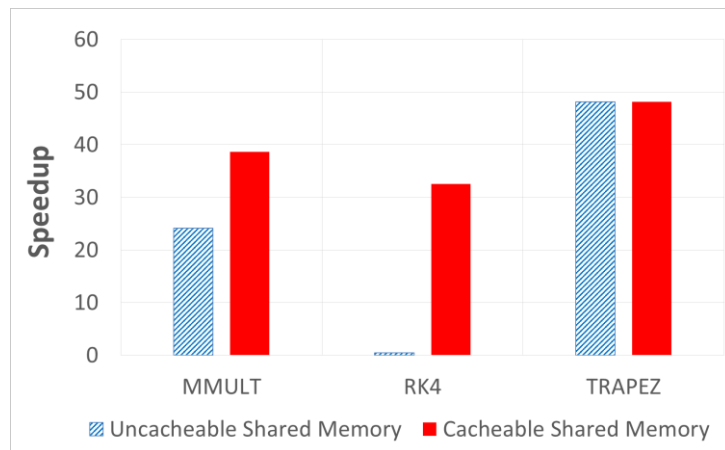
- 48-core experimental processor
- Private **non-coherent** caches
- On-chip Message Passing Buffer (MPB)
- Shared off-chip main memory - **Uncacheable**
 - No locking!



From TFlux to SWitches - TFluxSCC[‡]

[‡]Diavastos, Andreas, Giannos Stylianou, and Pedro Trancoso. "TFluxSCC: Exploiting Performance on Future Many-Core Systems through Data-Flow." *Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference on*. IEEE, 2015.

- TFluxSCC Memory model
 - MPB for TSU updates
 - Shared off-chip for application data



Application
Data

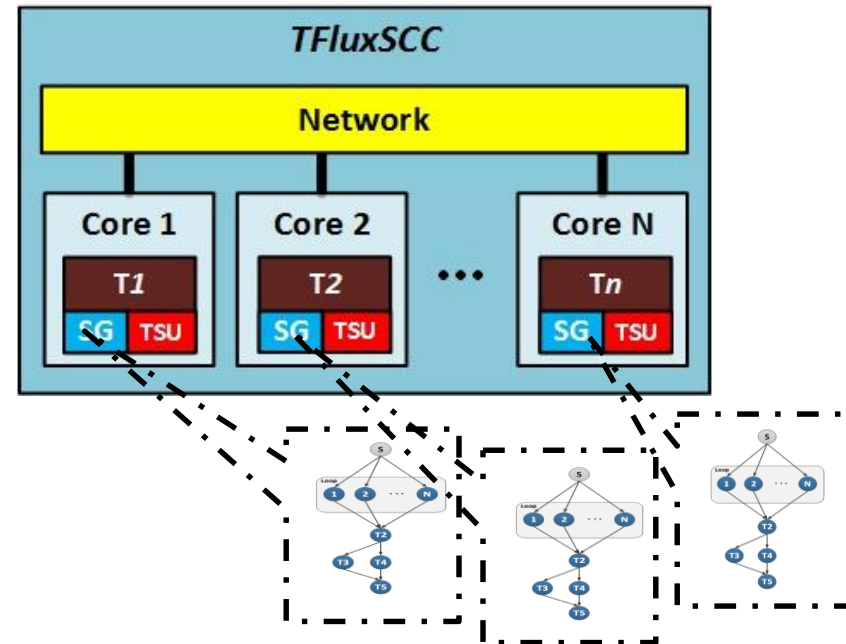
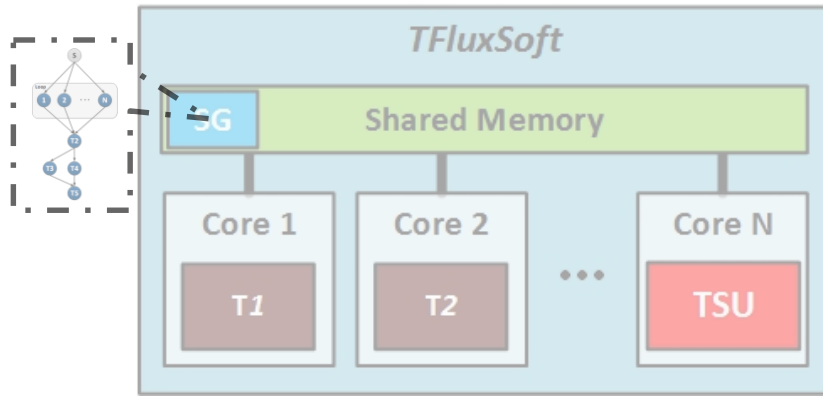
TSU
Updates

- Simultaneous access is not allowed on shared data in DDM
- Caching global data is enabled in TFluxSCC
- Flush caches to ensure write-back is complete



TFluxSCC[‡] Runtime Implementation

[‡]Diavastos, Andreas, Giannos Stylianou, and Pedro Trancoso. "TFluxSCC: Exploiting Performance on Future Many-Core Systems through Data-Flow." *Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference on.* IEEE, 2015.

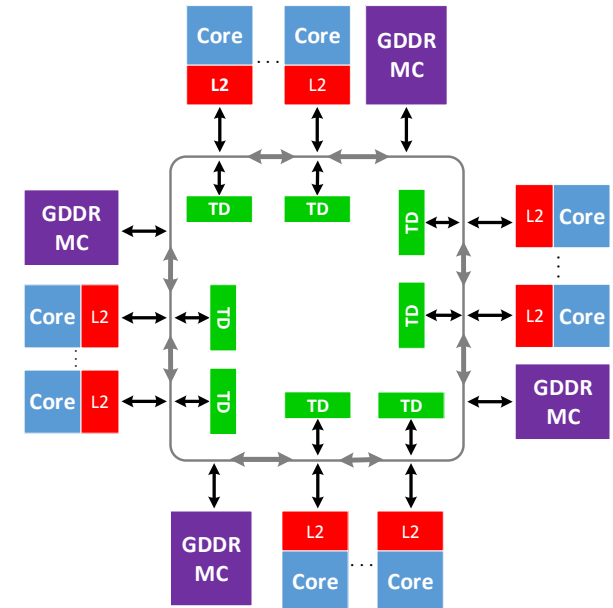


- Fully **decentralized runtime** system
- One TSU on every application thread
 - Multiple communication points
- Each core has its own instance of SG
 - No locking



TFluxSoft on Intel Xeon Phi[‡]

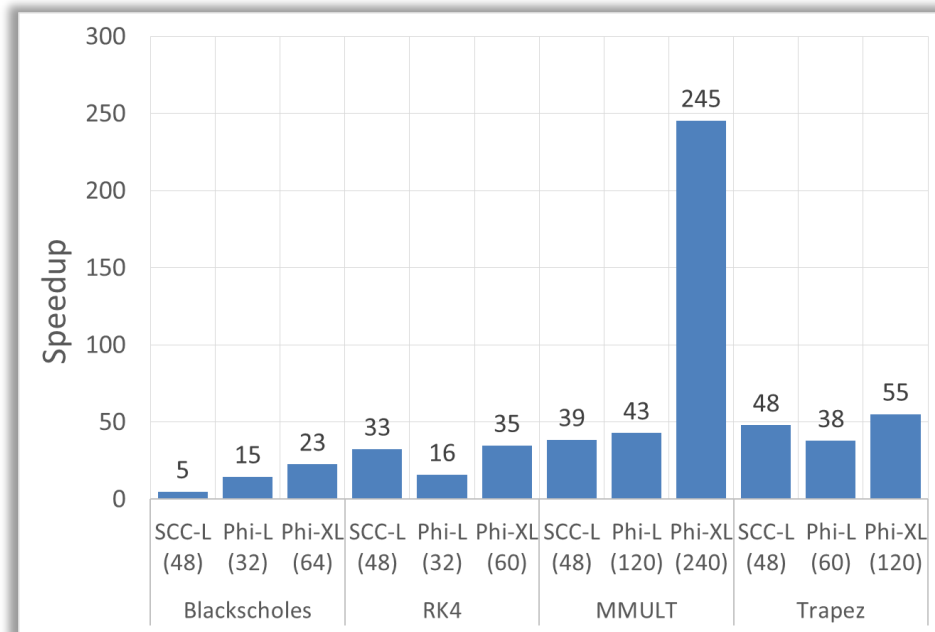
- Intel Many Integrated Core Architecture
 - Knights Corner series
- 61 cores interconnected with on-die bidirectional ring
- 4-threaded cores (totaling 244 hardware threads)
- X86 ISA with 64-bit addressing
- 32 MB LLC – **shared, cache-coherent**
 - 512KB L2 per core



[‡] We would like to thank The Cyprus Institute for letting use their Xeon Phi cluster for our experiments



Results: TFluxSCC & TFluxSoft



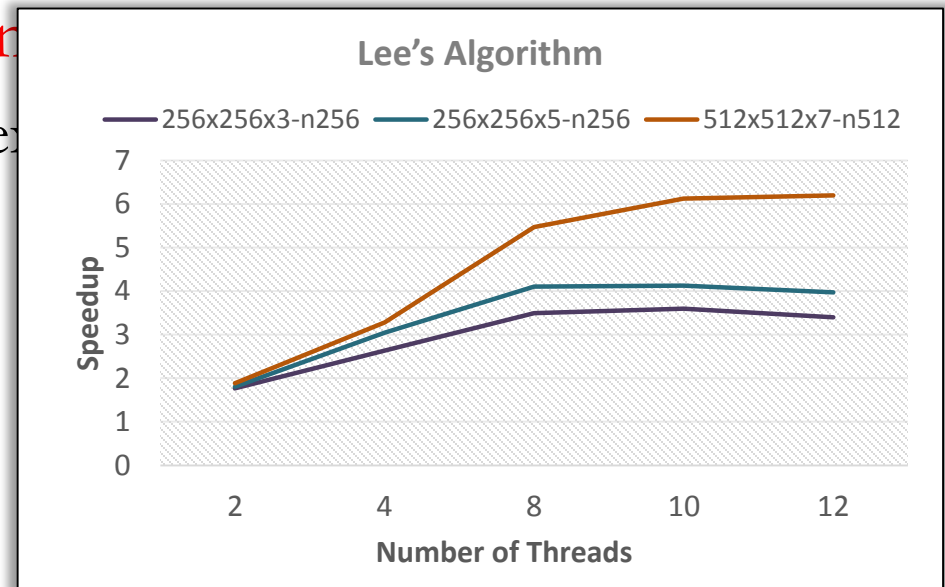
- **TFluxSCC:** Distributed TSU and local-SG
 - Multiple communication points for better scalability
 - Avoid monitoring & locking of shared data
- **TFluxSoft:** Centralized TSU and global-SG
 - Single point of synchronization
 - Contention on shared SG by all executing thread



From TFlux to SWitches - TFluxTM[‡]

[‡]A. Diavastos, P. Trancoso, M. Lujan and I. Watson, "Integrating Transactions into the Data-Driven Multi-threading Model Using the TFlux Platform," *International Journal of Parallel Programming*, pp. 1-21, 2015

- Integrated Transactional Memory support
 - Flexibility of thread-based DDM
- Provide atomic operations
- Exploit runtime parallelism
 - Using speculative parallel execution
- Support more applications



Conclusions of TFlux experience

- **Positives:**
 - Software implementation
 - Portable across commodity hardware systems
 - Thread-based granularity
 - Apply architectural & compiler optimization within threads
 - Pipelining, unrolling, vectorization, etc.
 - Flexibility of DDM
 - Integrate multiple models in one system
 - Shared-Memory model
 - **Makes programming easier!**
 - All-the-time coherent execution
 - Data dependencies prohibit simultaneous access on shared data
 - **No cache-coherence needed!!**



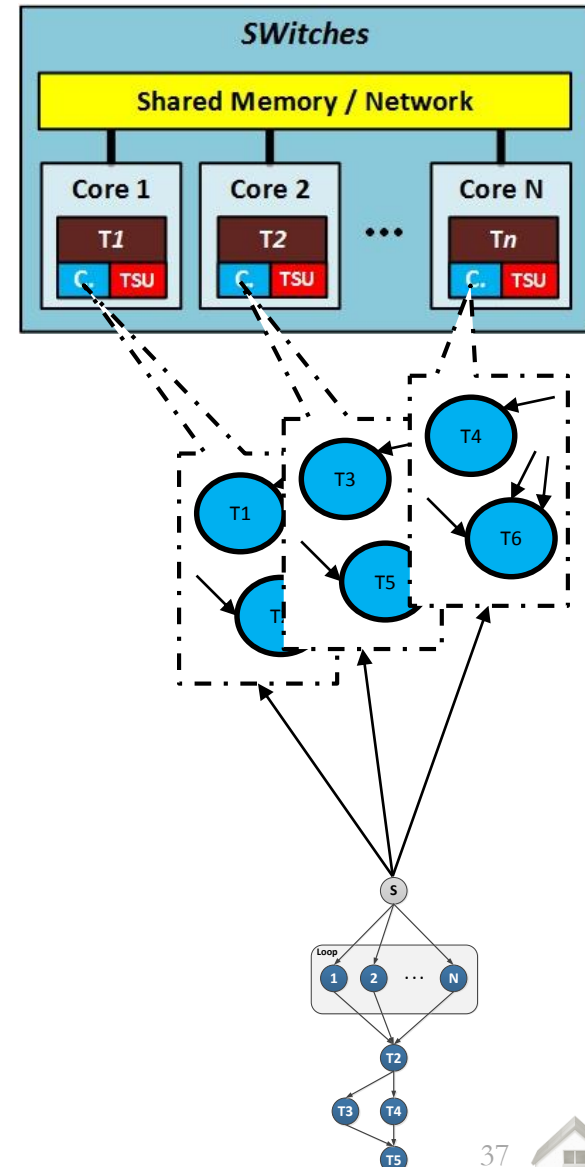
Conclusions of TFlux experience

- **Negatives:**
 - Large TSU structures (SG)
 - High memory footprint
 - Large initialization time
 - 65% of total execution time is used for TSU allocation & initialization
 - Shared SG
 - Locking & Hardware coherence support
 - Centralized TSU
 - Single-point of communication
 - 30% of time is spent in the TSU
 - This will increase with the number of cores
 - Most of the time it's not useful operations

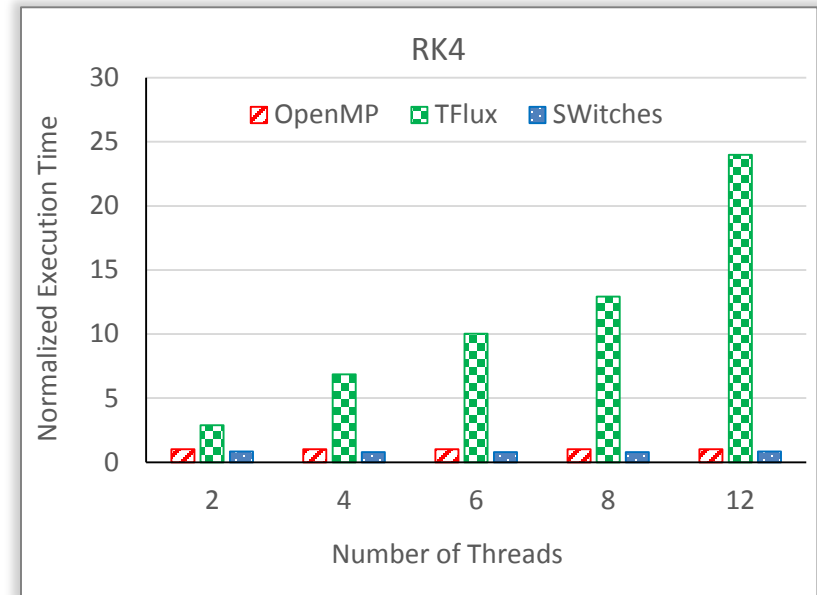
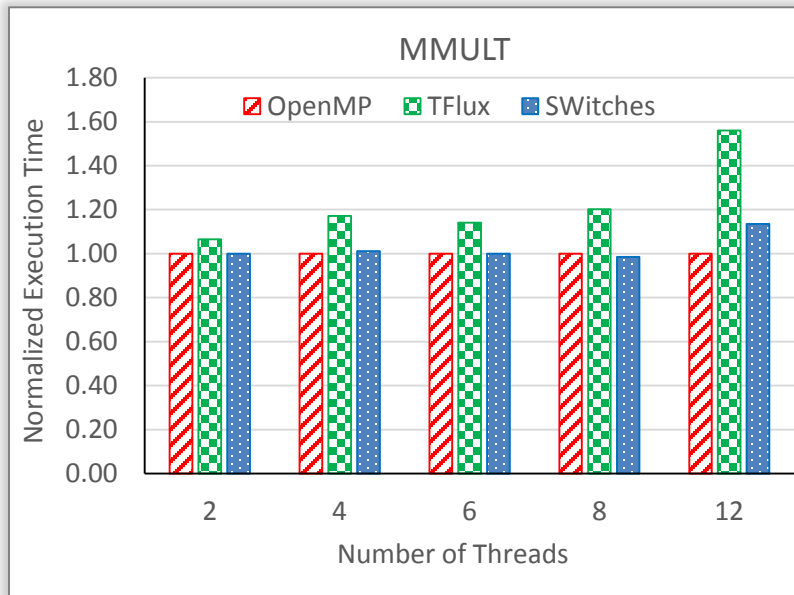


Switches: Key Characteristics

- Connect Producers – Consumers with switches
 - A unique switch for every producer
 - A producer turns ON it's switches when completed
 - A consumer starts when all it's producers switches are ON
 - Switches are boolean C/C++ data types
- **Decentralized SG & Runtime**
 - Single writer / Multiple readers (for every switch)
 - No locking needed
 - Low overhead notifications
- Use shared-memory
 - **No cache-coherence support needed!**



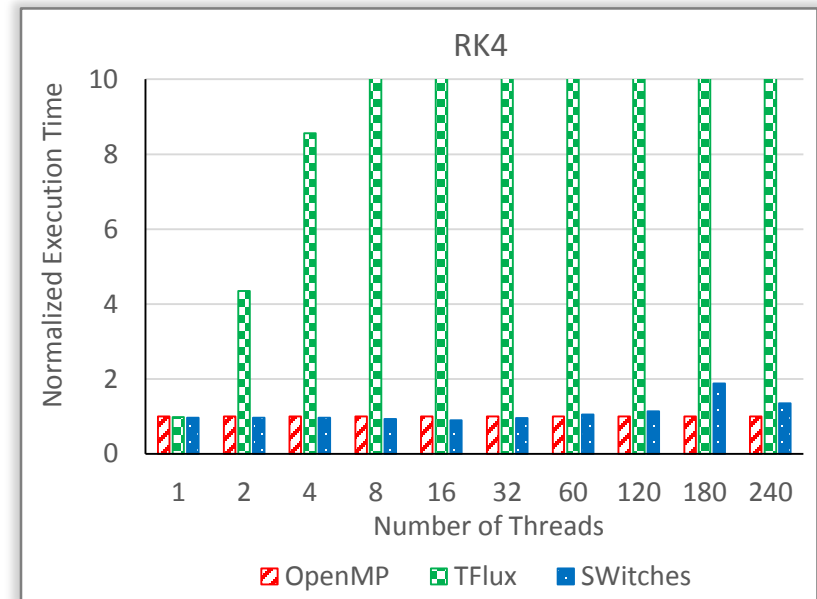
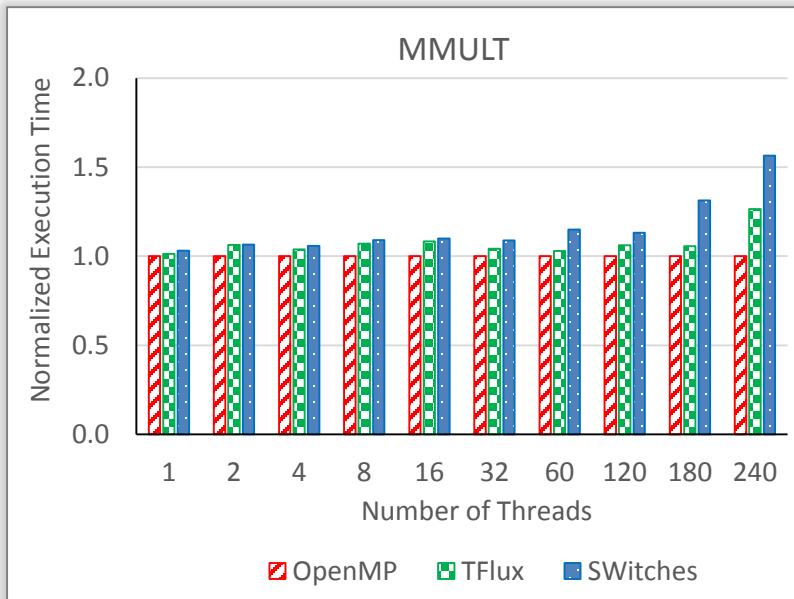
SWitches Evaluation: 12-core multicore



- **MMULT:** Good scalability
 - TFlux loses performance at 12 cores (+1 TSU thread)
- **RK4:** TFlux loses performance
 - TFlux improves execution up to 8 cores
 - Too many TFlux threads
 - Too large SG shared among threads



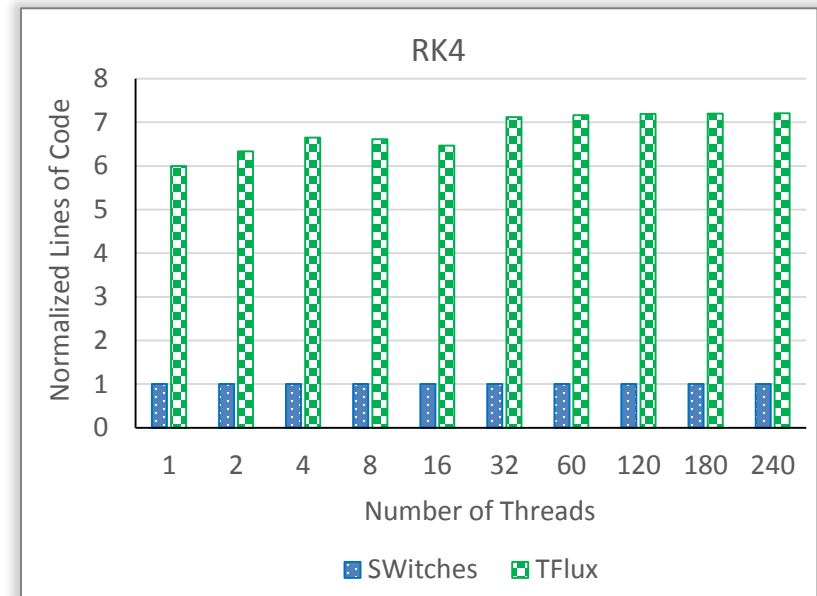
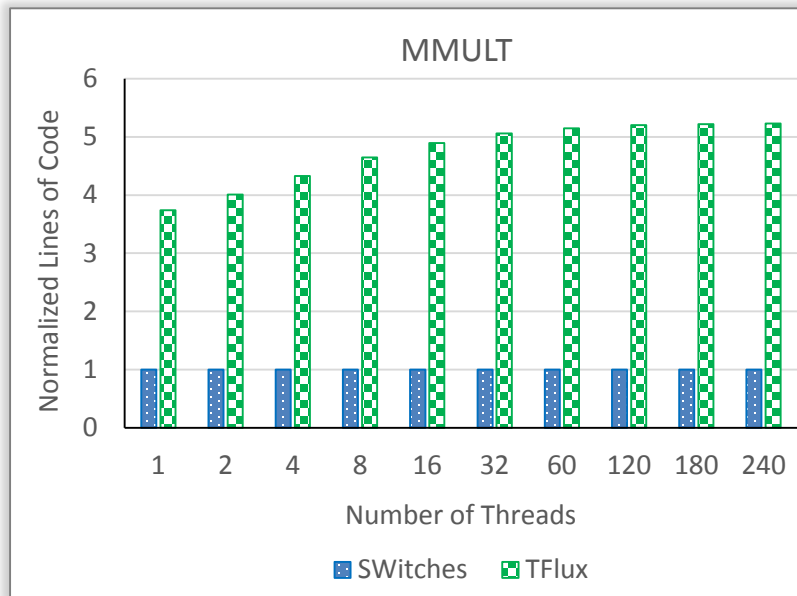
Switches Evaluation: 60-core Xeon Phi



- **MMULT:** Good scalability
 - SWitches is slightly slower than OpenMP & TFlux (difference is negligible)
- **RK4:** TFlux is $\approx 30\times$ slower
 - Centralized TSU becomes a communication bottleneck
 - Locking global SG with 240 threads is expensive!



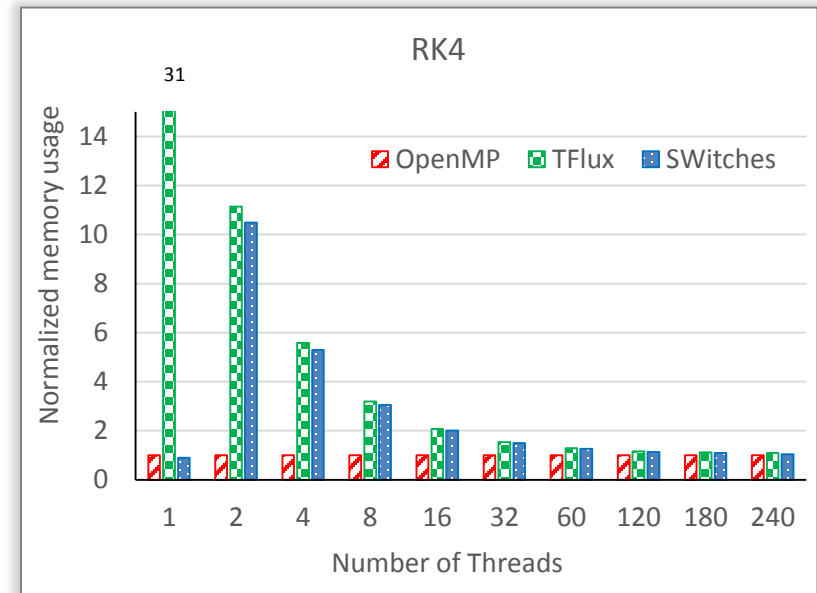
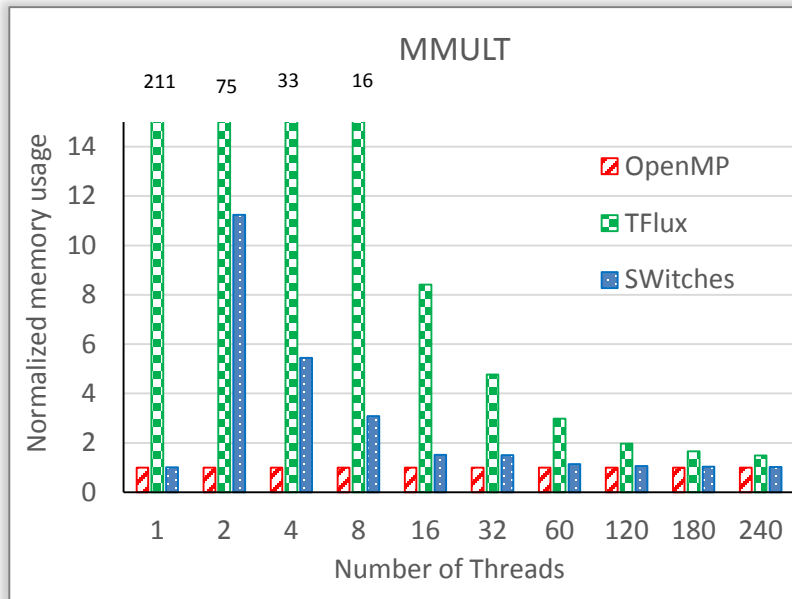
Switches Profiling: Lines of Code



- TFlux produces on average 6× more code
 - Calls to the runtime system
 - **More code to execute → More execution time**



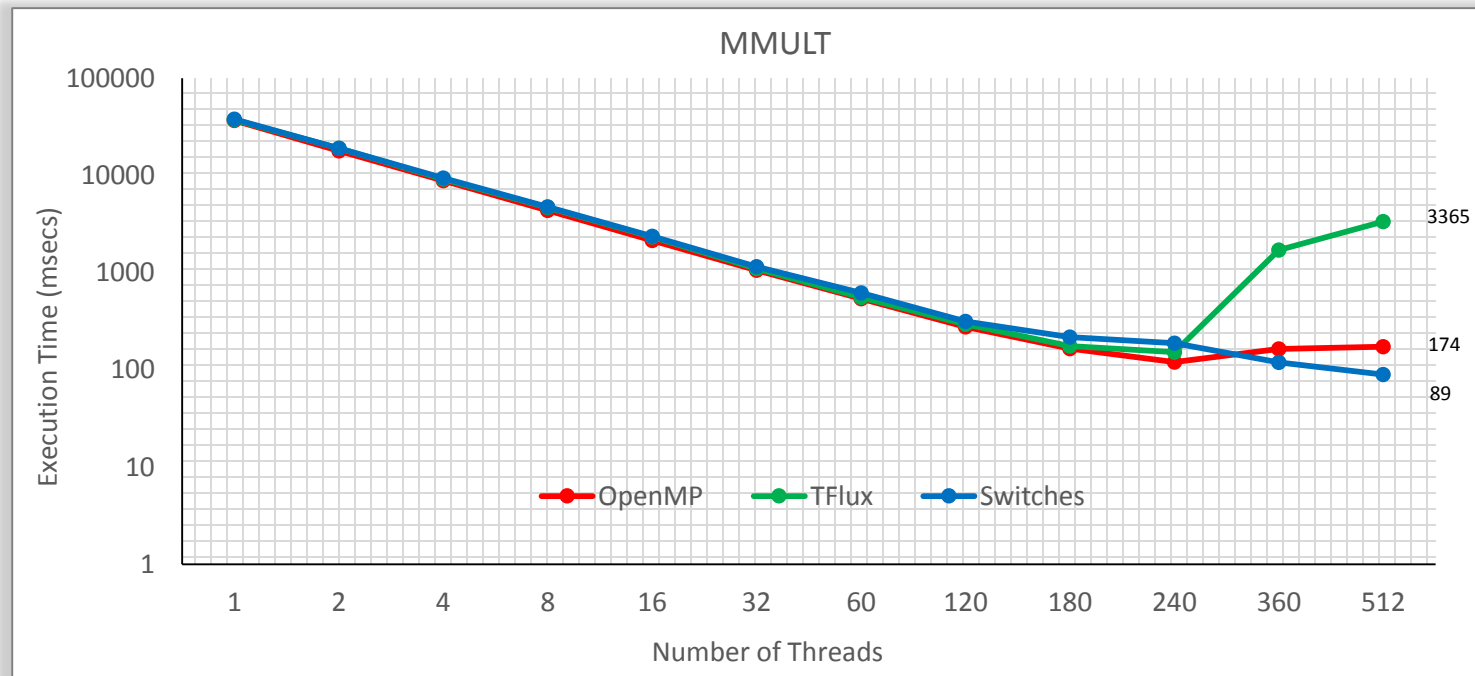
Switches Profiling: Memory Usage



- The more threads we use the more memory each system needs
- TFlux on average requires more memory for SG
 - SG is created regardless of the # of cores used
 - SWitches creates SG info using the # of cores used (less cores → less info)
 - Switches are less memory demanding (boolean)
 - OpenMP & SWitches use a **chunk-based technique to increase/decrease granularity** & reduce re-scheduling overheads



Switches Profiling: Scale beyond...



- **TFlux:** Locking of global SG & sharing among 512 threads!
- **TFlux:** Context switching interferes with centralized TSU (updates delayed)
- **OpenMP:** Also uses a centralized runtime
- **SWitches:** Decentralized runtime & local SG!!





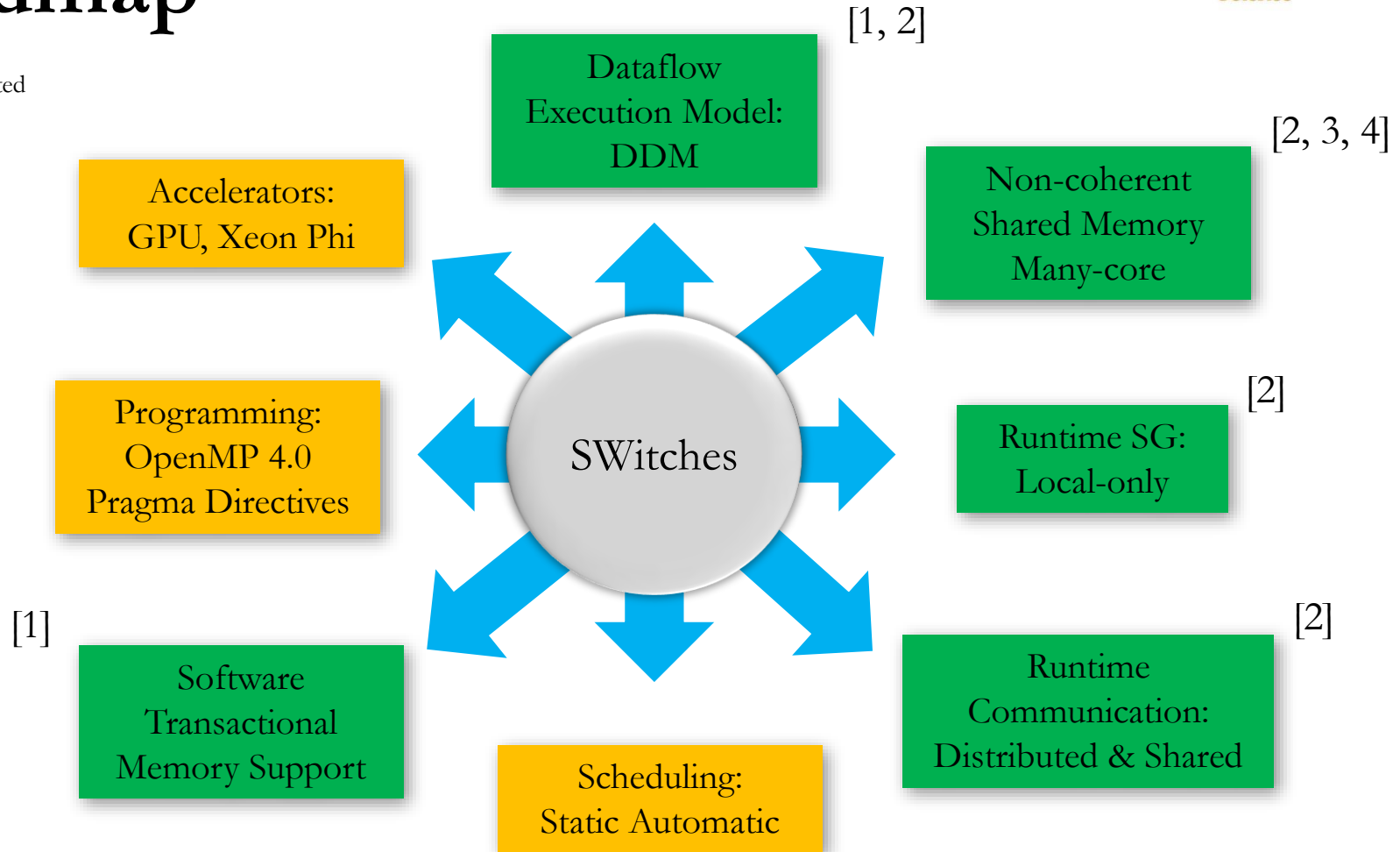
Outline

- Motivation
- Contributions
- Related Work
 - Data-flow-based Programming Systems
 - DDM Programming Systems
- SWitches Programming System
- Preliminary Work
- Roadmap – Timeline



Roadmap

 Completed
 Pending



[1] **Andreas Diavastos**, Pedro Trancoso, Mikel Lujan and Ian Watson. “Integrating Transactions into the Data-Driven Multi-threading Model using the TFlux Platform”, International Journal of Parallel Programming (IJPP 2015).

[2] **Andreas Diavastos**, Giannos Stylianou and Pedro Trancoso “TFluxSCC: Exploiting Performance on Future Many-core Systems through Data-Flow”. In the Proceedings of the 23rd Euromicro International Conference, Distributed and Network-based Processing (PDP 2015).

[3] Panayiotis Petrides, **Andreas Diavastos**, Constantinos Christofi and Pedro Trancoso. “Scalability and Efficiency of Database Queries on Future Many-core Systems”. In the Proceedings of the 21st Euromicro International Conference, Distributed and Network-based Processing (PDP 2013).

[4] **Andreas Diavastos**, Panayiotis Petrides, Gabriel Falcao, Pedro Trancoso. “LDPC Decoding on the Intel SCC”, In the Proceedings of the 20th Euromicro International Conference, Distributed and Network-based Processing (PDP 2012).

Thank You!



University of Cyprus
Department of Computer
Science

CASPER Group

Visit Us: www.cs.ucy.ac.cy/carch/casper

Computer Architecture, Systems and Performance Evaluation Research

List of Publications:

Journals:

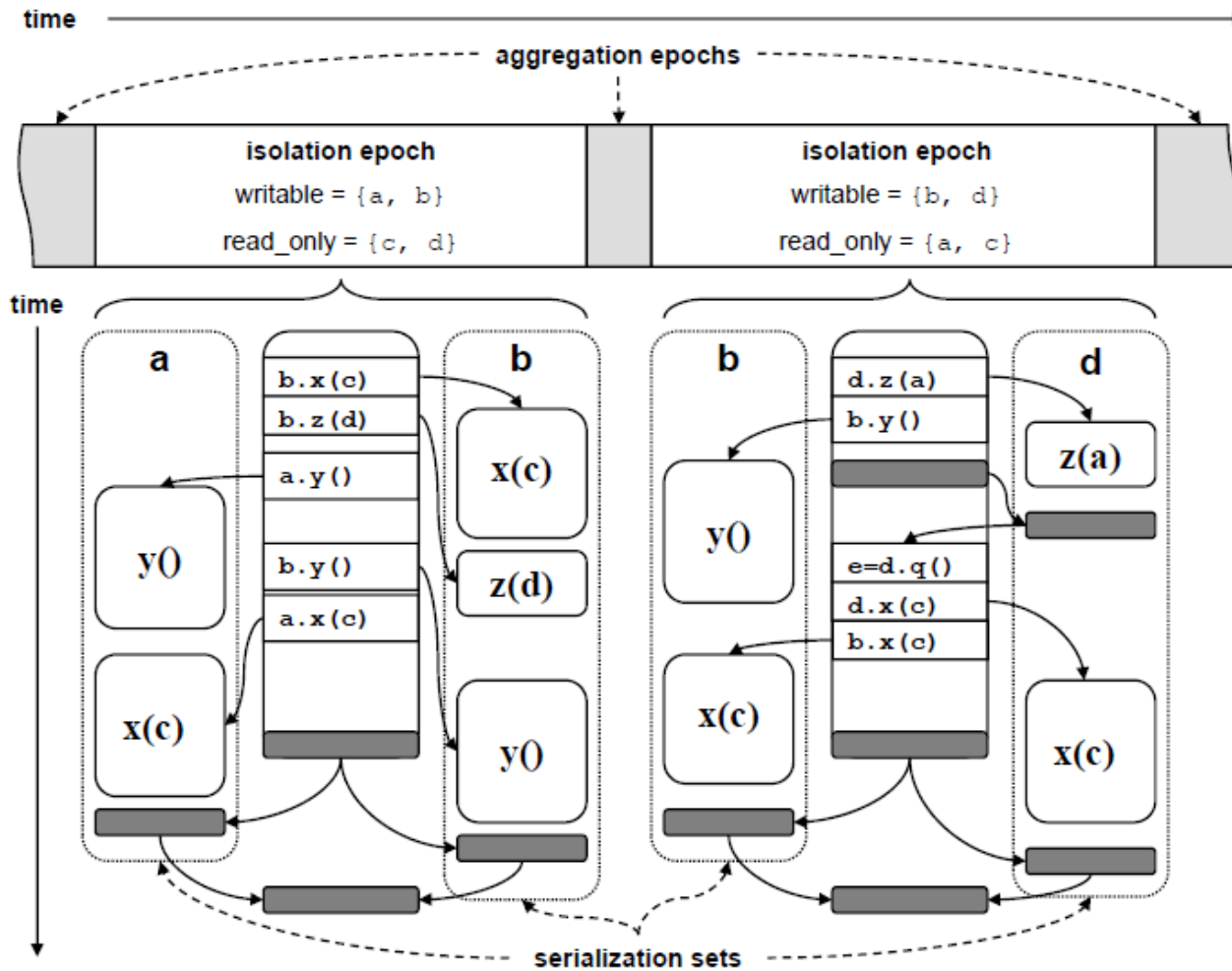
1. **Andreas Diavastos**, Pedro Trancoso, Mikel Lujan and Ian Watson. “*Integrating Transactions into the Data-Driven Multi-threading Model using the TFlux Platform*”, International Journal of Parallel Programming (IJPP 2015).
2. **Andreas Diavastos**, Giannos Stylianou, Giannis Koutsou. “*Exploiting Very-Wide Vector Processing for Scientific Applications*”. (Article) In Computing in Science & Engineering (CiSE), Vol. 17, no. 6, Nov/Dec 2015, pp. 83.87.

Conferences & Workshops:

3. **Andreas Diavastos**, Giannos Stylianou and Pedro Trancoso “*TFluxSCC: Exploiting Performance on Future Many-core Systems through Data-Flow*”. In the Proceedings of the 23rd Euromicro International Conference, Distributed and Network-based Processing (PDP 2015).
4. **Andreas Diavastos**, Giannos Stylianou, Pedro Trancoso. “*TFluxSCC: A Case Study for Exploiting Performance in Future Many-core Systems*”. (Poster Paper) In the Proceedings of the 11th ACM Conference on Computing Frontiers, Cagliari, Italy, May 2014.
5. Panayiotis Petrides, **Andreas Diavastos**, Constantinos Christofi and Pedro Trancoso. “*Scalability and Efficiency of Database Queries on Future Many-core Systems*”. In the Proceedings of the 21st Euromicro International Conference, Distributed and Network-based Processing (PDP 2013).
6. **Andreas Diavastos**, Panayiotis Petrides, Gabriel Falcao, Pedro Trancoso. “*LDPC Decoding on the Intel SCC*”, In the Proceedings of the 20th Euromicro International Conference, Distributed and Network-based Processing (PDP 2012).
7. **Andreas Diavastos**, Pedro Trancoso, Mikel Lujan and Ian Watson. “*Integrating Transactions into the Data-Driven Multi-threading Model using the TFlux Platform*”, In the Proceedings of the Data-Flow Execution Models for Extreme Scale Computing Workshop (DFM 2011).
8. **Andreas Diavastos**, Giannos Stylianou and Giannis Koutsou “*Exploiting Very-Wide Vectors on Intel Xeon Phi with Lattice-QCD kernels*”. In the Proceedings of the 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2016).

Backup Slides

Serialization Sets

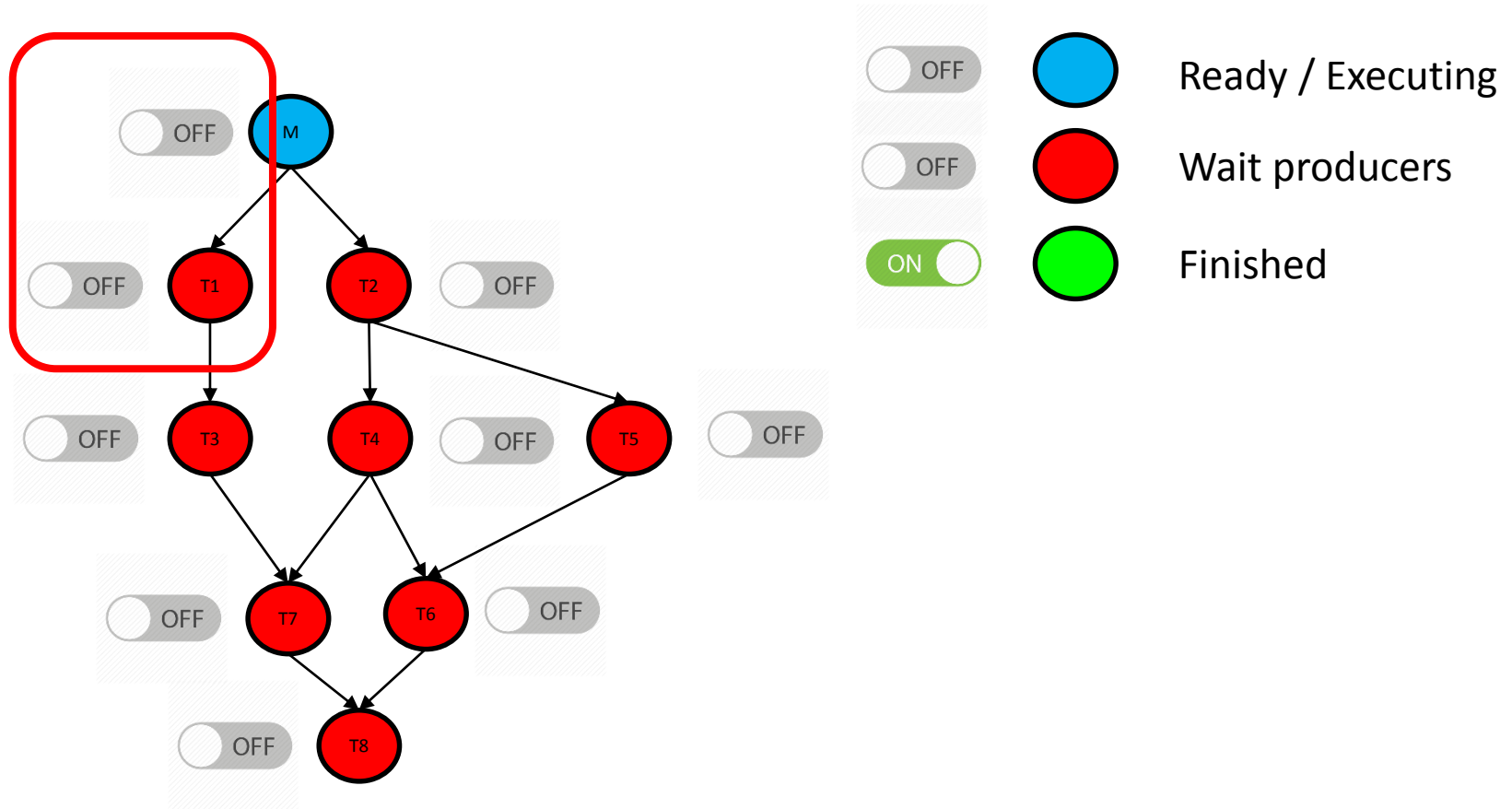


Future Direction of Work

- Implement & test more applications
- Parallel threads acceleration
 - GPUs
 - FPGAs
- **Test SWitches on HPC systems**
 - In collaboration with message-passing systems
 - Performance scalability on multi-node systems
 - e.g. SWitches for intra-node & MPI for inter-node
 - Test real HPC scientific applications



Example execution of SWitches



Example execution of SWitches

